ARL-TR-7589 ● JAN 2016

US Army Research Laboratory

Touch-Based Interaction Approach for Network Science Research and Visualization

by John P Hancock, Mathew Aguirre, and Andrew Toth

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7589 ● JAN 2016

US Army Research Laboratory

Touch-Based Interaction Approach for Network Science Research and Visualization

by John P Hancock and Mathew Aguirre
ArtisTech, Inc.

Andrew Toth
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2016	Final	10/2014–09/2015

4. TITLE AND SUBTITLE
Touch-Based Interaction Approach for Network Science Research and Visualization

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John P Hancock, Mathew Aguirre, and Andrew Toth

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-T
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7589

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ARL TNAB

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Visualization of network science experimentation results is generally achieved using stovepipe solutions tailored to specific experiments and performance metrics. Based on ZeroMQ, the US Army Research Laboratory (ARL) Visualization Framework presents a language-agnostic, platform-independent approach to connecting data published by probes to visualizations using a publish/subscribe mechanism. Visualizations can be augmented using touch-enabled displays such as the MultiTaction M550L. This report documents adaptation of the MultiTaction display to the ARL Visualization Framework using a Tangible User Interface Object (TUIO).

15. SUBJECT TERMS
TUIO, Visualization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Andrew Toth
Unclassified	Unclassified	Unclassified	UU	22	19b. TELEPHONE NUMBER (Include area code) 301-394-2746

Contents

List of Figures

1. Introduction

The US Army Research Laboratory (ARL) Network Science Research Laboratory (NSRL) is composed of a suite of hardware and software that models the operation of mobile networked device radio frequency (RF) links through emulation (not merely simulation). NSRL enables experimental validation or falsification of theoretical models, and characterization of protocols and algorithms for mobile wireless networks. It is used for a range of experiments, from assessing in-network aggregation of network information for detecting cyber threats, to characterizing the impact of communications disruption on perceived trust and quality of information metrics delivered to Soldiers in tactical mobile environments. Unlike other experimentation facilities for research in wireless networks, NSRL is focused on Army-unique requirements like hybrid networks and extensive modeling of ground and urban effects on communications. The NSRL supports investigation of traditional wireless networking challenges, as well as more general network science research issues. The NSRL's emulation environment is result of collaborative efforts between ARL and the Naval Research Laboratory (NRL).

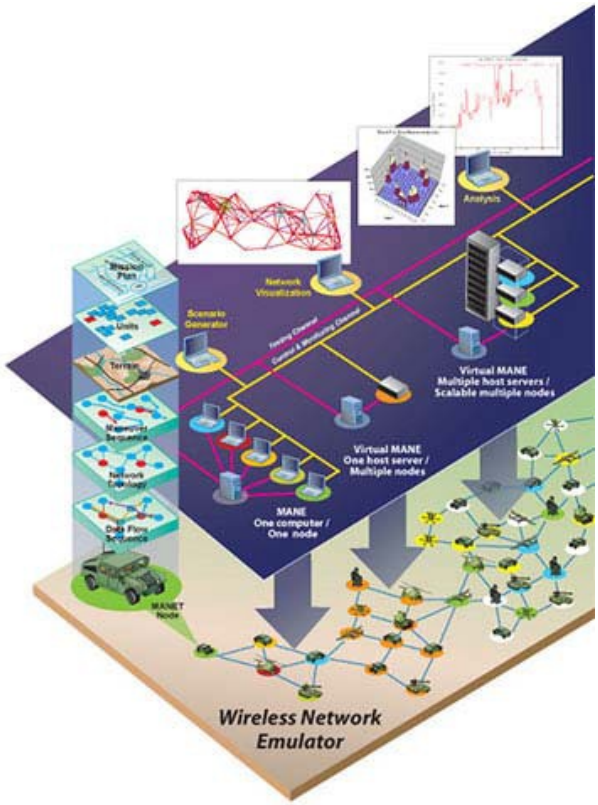

Fig. 1 NSRL Experimentation Architecture

The software and systems running on the emulated networks execute in real-time, unlike simulations, which typically execute faster than real-time by jumping from event to event, skipping the time between events. One of the biggest advantages of emulated environments is that the characteristics of the network and its behaviors can be fully controlled and are repeatable. This is particularly interesting when emulating wireless networks, as reproducibility of experimental environments using real radios is often difficult; temperature and humidity changes, differences in seasonal foliage, and other factors can alter the performance of the wireless networks. The primary emulation tools used by ARL are the Extendable Mobile Ad hoc Network Emulator (EMANE)[1] and the Common Open Research Emulator CORE.[2]

Researchers at the NSRL developed the ARL Visualization Framework[3] to improve the process of network science experimentation by providing a generic approach to experimental results visualization. Most experiments collect data via log files, which are written to a storage device for post-processing and analysis. Experimental results in the individual log files must be synchronized prior to processing to link data points of individual experiment runs. Visualization of results has typically been accomplished using "stove-pipe" visualization applications developed specifically for a particular experiment.

The ARL Visualization Framework will use a ZeroMQ-based bus to route data from Probes to VizDaemons interested in displaying the probe data using a publish/subscribe architecture. Probes can be any application or script that can produce ZeroMQ-compliant messages containing data of interest. This data can range from network metrics collected in an experiment to server CPU statistics. VizDaemons consume only the ZeroMQ messages published by the probes to which they subscribe and display them accordingly. While VizDaemons will typically be used for visual output types, there is no reason the output cannot be aural. A simplified depiction of the ARL Visualization Framework can be seen in Fig. 2.

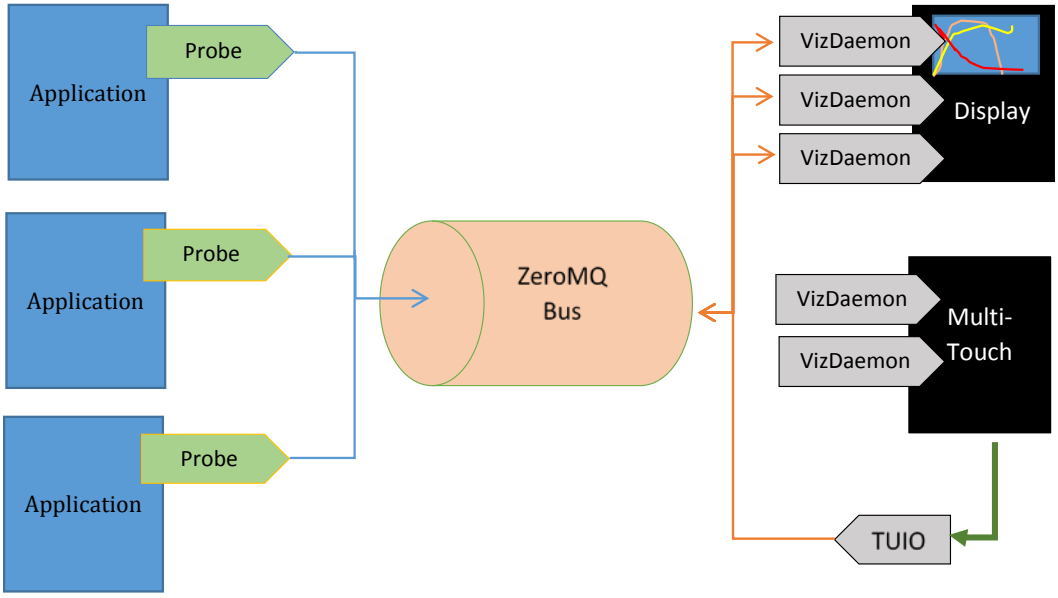

Fig. 2 ARL Visualization Framework Simplified Architecture

This ARL technical report documents the design of a reusable interface for multi-touch interactive displays in the NSRL that leverages the ARL Visualization Framework. The multi-touch display from MultiTaction[4] allows multiple users to interact with the display simultaneously (Fig. 3) in several advanced touch interaction metaphors. An interface approach that uses open source libraries, which allows mapping of simple and complex multi-user interactions to a visualization server, is required to distribute interaction notifications to multiple subscribed systems in the NSRL.

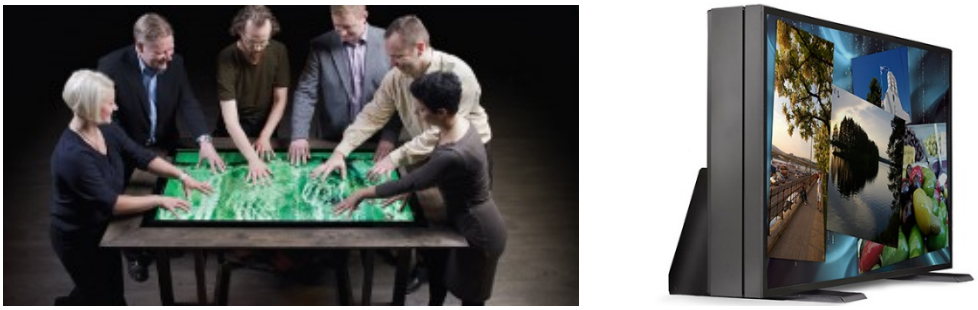

Fig. 3 MultiTaction display device and multi-user tactile interaction

This approach will allow multiple users to interact with multiple NSRL applications through a shared interface. The interface also goes beyond point/drag/click mouse metaphors, allowing hand and gesture recognition to interact with objects and blobs. Mapping of these gesture-defined elements will be done as the NSRL functions come online and are ready for visualization. This work paves the way for future advanced interface design by bringing the touchscreen interaction detections

out to a server that can interact with other subscribed NSRL display and computing systems.

The NSRL currently has 2 MultiTaction 550L embeddable displays. Each of these displays has an internal Linux server that reads, interprets, tracks, and reports multiple simultaneous human touch events on the screen surface. The display computer can be configured to send these interpreted events out in an open standard Tangible User Interface Object (TUIO)[5,6] format, which has cursor, object, and blob definitions. In this work we defined an architecture, developed a driver, and designed and developed a server to distribute reported screen interactions to any number of subscribed applications in the NSRL. The TUIO interaction architecture is illustrated in Fig. 4.

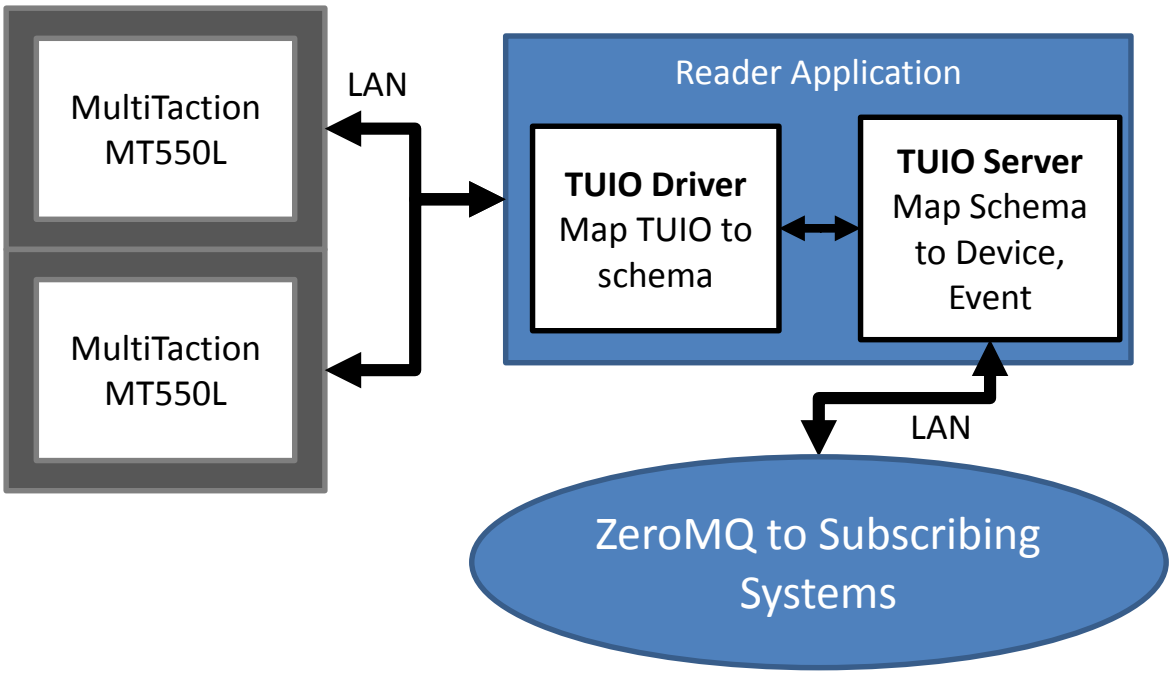

Fig. 4 TUIO Interaction Architecture

We used a TUIO-based approach to integrate ARL's Multi-Taction (MT) MT-550L displays to the visualization capabilities that are emerging for the new NSRL. The 2-dimensional (2D) TUIO network-based interfaces were mapped to the MT550L to work with both Linux and Mac operating systems. This solution was created using an open source approach tailored to the specific needs of ARL. This approach is general and should apply to other touch screen devices, and be extensible to objects, blobs, and even haptic or other 3D interfaces in the future.

The system diagram of the Pub/Sub architecture in Fig. 4 illustrates the high-level view of the functional and software architecture of the visualization interaction system. The TUIO Driver and Server are both written in Java with interfaces

defined using the Google Protocol Buffers. The server has a protocol buffer definition of the 2D TUIO interfaces; the full set (as of this writing) is in Appendix, and the latest is always available at https://github.com/artistech-inc/tuio-zeromq-publish/blob/master/src/main/resources/protobufs/TUIO.proto

The MultiTaction displays are configured to output captured tactile events in TUIO format over the attached Ethernet. The TUIO reader application reads the tactile TUIO events from each screen, maps them to an event model schema, and sends them to be distributed to subscribed systems in the NSRL via a ZeroMQ message queue server.

The TUIO interfaces are based on the open source definitional specification available online at the open source website[8] for the standard.

This Java-based approach is designed to support both the Linux and Mac OS machines in the NSRL, and has also been tested on Android devices.

2. TUIO Mouse Driver

The TUIO Driver was configured, for this initial prototype, as a mouse driver to run on devices that have their interfaces displayed on the MT displays. Mouse move, left-click, and right-click are implemented, and the MT display was tested as the touch interface for a Mac computer browser on the NSRL network.

The following motions are how a user manipulates the touch screen acting as a mouse using a TUIO device.

1) One finger is for mouse move.
2) Two fingers is for left-click. Mouse down is maintained while the second finger is pressed.
3) Three fingers is right-click. Mouse up is immediately called; no need to keep finger down.

The TUIO Mouse Driver cursor control is the first tactile message interface that was developed, integrated, and successfully tested on the ARL MT displays. The TUIO Server message schema is implemented in the Google Protocol Buffer interface definition detailed below describing available fields.

```
message Cursor {
        required float x = 1;
        required float y = 2;
        optional Time tuioTime = 3;
        optional Time startTime = 4;
```

```
        required int64 sessionID = 5;
        optional float xSpeed = 6;
        optional float ySpeed = 7;
        optional float motionSpeed = 8;
        optional float motionAccel = 9;
        repeated Point path = 10;
        required int32 tuioState = 11;
        required int32 cursorID = 12;
}
```

2.1 Installation

The mouse driver developed for this program is hosted on github, which is a web-based git repository.[7] To get and make the mouse driver:

1) git clone https://github.com/artistech-inc/tuio-mouse-driver.git
2) cd tuio-mouse-driver
3) git checkout v1.1.3
4) mvn package
5) java -jar target/tuio-mouse-driver-1.1.3.jar

2.2 ZeroMQ Support

This module supports the ability to subscribe to TUIO broadcasts via Zero Message Queue, ZeroMQ, to integrate with the ARL Visualization Framework.

2.3 Dependencies

ZeroMQ support is dependent on available native libraries. When compiling, maven will search for these files and provide any jar dependencies suitable.

1) Linux:
    1) Searches for /usr/lib/libjzmq.so
    2) If this file exists, the dependency jar jzmq.jar is imported.
    3) If this file is missing, the dependency jar jeromq.jar is imported.
2) Mac OS X:
    1) Searches for /usr/lib/libjzmq.dynlib
    2) If this file exists, the dependency jar jzmq.jar is imported.
    3) If this file is missing, the dependency jar jeromq.jar is imported.

The 2 jar files provide identical support. However, the jzmq.jar file uses JNI to provide faster support, where jeromq.jar is a pure java implementation. The jzmq.jar requires libjzmq.so, which in turn requires libzmq.so to be available.

2.4 ZeroMQ Transmission/Serialization

For the purposes of NSRL visualization, the decision was made to standardize on Google Protocol Buffer interface definitions, but in the larger scheme of Network Science experimentation, other interaction serialization mechanisms are also used. We took the inclusive approach to design of the Zero Message Queue publish and subscribe service, and included multiple serialization approaches. Transmission of the TUIO messages via ZeroMQ is provided by 3 different mechanisms:

1) Java Object Serialization
2) JSON Serialization (using Jackson)
3) Google Protocol Buffer

In this open architecture, to the ZeroMQ subscribing client it is unknown how an incoming object has been serialized by the publisher, and so all 3 mechanisms are attempted for deserialization. This approach generalizes the interface:

1) For any clients that are doing Java and want to serialize, and are not using Google Protocol Buffers as the intermediary.
2) For any other language that wants to interact but is not using Google Protocol Buffers as the intermediary.
3) For any language that *is* using Google Protocol Buffers.

This open interface approach will simplify integration with a wide range of applications using Java objects, JSON, or Google Protocol Buffers.

2.5 Installation

To get and make the ZeroMqMouse driver:

1) git clone https://github.com/artistech-inc/tuio-mouse-driver.git
2) cd tuio-mouse-driver
3) git checkout v1.1.3
4) mvn package

5) java -cp target/tuio-mouse-driver-1.1.3.jar com.artistech.tuio.mouse.ZeroMqMouse -z <ZMQ_PUB_HOST:PORT>

If using the companion tuio-zeromq-publish application, the default port used is 5565, so invocation would look similar to:

java -cp target/tuio-mouse-driver-1.1.3.jar com.artistech.tuio.mouse.ZeroMqMouse -z localhost:5565

The latest Mouse Driver technical documentation and code is maintained at: https://github.com/artistech-inc/tuio-mouse-driver, the following is a version of it that is current at the writing of this document.

3. TUIO ZeroMQ Publish

Integrating the Multi-Taction displays with the ARL Visualization Framework requires a publish bridge for receiving TUIO messages and then publishing them via ZeroMQ to the visualization framework. This section describes the publish bridge.

3.1 Installation

To get and make the TUIO publisher:

1) git clone https://github.com/artistech-inc/tuio-zeromq-publish.git
2) cd tuio-zeromq-publish
3) git checkout v1.1
4) mvn package
5) java -jar target/tuio-zeromq-publish-1.1.jar

As configured, this application will listen (by default) on port 3333 for TUIO messages. Once received, these messages are serialized (by default) using Google Protocol Buffers and published (by default) on port 5565. A companion client can be used to receive these messages, deserialize, and process.

To change these options the following command line options are available:

```
-h,--help                       Show this message.
-s,--serialize-method <arg>     Serialization Method (JSON, OBJECT,
Default
                                  = PROTOBUF).
-t,--tuio-port <arg>            TUIO Port to listen on. (Default =
3333)
```

```
-z,--zeromq-port <arg>        ZeroMQ Port to publish on. (Default
= 5565)
```

3.2 ZeroMQ Transmission/Serialization

Transmission of the TUIO objects via ZeroMQ is provided by 3 different mechanisms.

1) Java Object Serialization
2) JSON Serialization (using Jackson)
3) Google Protocol Buffer

The ZeroMQ Publish technical documentation is available at the ArtisTech github site;[7] the following is a version that is current at the writing of this document. The applications, code, and documentation are part of an ongoing program to conduct experiments in the Network Science CTA, generally, and in the new NSRL, specifically. This TUIO display, control, and dissemination system is intended for use by ARL and associated researchers, who we expect will have variant needs and will evolve this set of applications to meet unique visualization and control needs. GitHub provides a clean public mechanism to distribute, maintain, and if need be, fork the code and control maintainability.

3.3 Conclusion and Future Directions

Looking toward the future, the current implementation of the touch interface has only implemented the most familiar touch interactions. As seen Fig. 5, the MT displays have a richer set of multiple point touch, pen, and other interactions.

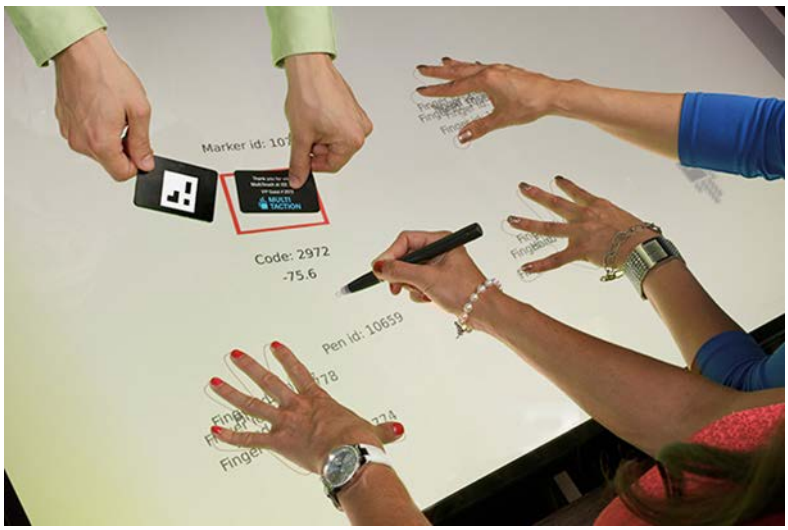

Fig. 5 MultiTaction display interaction methods

As Network Science matures as a topic, there will be opportunity to apply more complex interaction manipulations to displays, experiments, simulations and datasets; this TUIO-based approached is designed to be extensible and flexible to support a wide range of screen interaction metaphors.

4. References

1. Extendable Mobile Ad-hoc Network Emulator (EMANE), http://www.nrl.navy.mil/itd/ncs/products/emane
2. Common Open Research Emulator (CORE), http://www.nrl.navy.mil/itd/ncs/products/core
3. Dron Will, Keaton Mark, Hancock John, Aguirre Mathew, Toth Andrew J. US Army Research Laboratory visualization framework design document. Adelphi (MD): Army Research Laboratory (US); to be published. Report No.: ARL-TR-7561.
4. http://www.multitaction.com
5. Kaltenbrunner M, Bovermann T, Bencina R, Costanza E. TUIO A Protocol for Table-Top Tangible User Interfaces, (http://mtg.upf.edu/node/408)
6. http://www.tuio.org
7. https://github.com/artistech-inc
8. TUIO Specification: http://tuio.org/?specification

INTENTIONALLY LEFT BLANK.

Appendix. TUIO Protocol Buffer Message Definitions

```
package TUIO;

option java_package = "com.artistech.protobuf";
option java_outer_classname = "TuioProtos";

message Time {
    required int64 seconds = 1;
    required int64 microseconds = 2;
    required int64 frameID = 3;
}

message Cursor {

    required float x = 1;
    required float y = 2;
    optional Time tuioTime = 3;
    optional Time startTime = 4;

    required int64 sessionID = 5;
    optional float xSpeed = 6;
    optional float ySpeed = 7;
    optional float motionSpeed = 8;
    optional float motionAccel = 9;
    repeated Point path = 10;
    required int32 tuioState = 11;

    required int32 cursorID = 12;

}

message Blob {

    required float x = 1;
    required float y = 2;
    optional Time tuioTime = 3;
    optional Time startTime = 4;

    required int64 sessionID = 5;
    optional float xSpeed = 6;
    optional float ySpeed = 7;
    optional float motionSpeed = 8;
    optional float motionAccel = 9;
    repeated Point path = 10;
    required int32 tuioState = 11;

    required int32 blobID = 12;
    required float angle = 13;
    required float width = 14;
    required float height = 15;
    required float area = 16;
    required float rotationSpeed = 17;
    required float rotationAccel = 18;

}

message Object {
```

```
    required float x = 1;
    required float y = 2;
    optional Time tuioTime = 3;
    optional Time startTime = 4;

    required int64 sessionID = 5;
    optional float xSpeed = 6;
    optional float ySpeed = 7;
    optional float motionSpeed = 8;
    optional float motionAccel = 9;
    repeated Point path = 10;
    required int32 tuioState = 11;

    required int32 symbolID = 12;
    required float angle = 13;
    required float rotationSpeed = 14;
    required float rotationAccel = 15;

}

message Point {

    required float x = 1;
    required float y = 2;
    optional Time tuioTime = 3;
    optional Time startTime = 4;
}
```

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

2 (PDF) US ARMY RESEARCH LAB RDRL CIN T A TOTH B RIVERA